(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 905 618 B1

(12) EUROPEAN PATENT SPECIFICATION

(45) Date of publication and mention of the grant of the patent:
18.02.2009 Bulletin 2009/08

(51) Int Cl.:
G06F 9/46 *(2006.01)*

(21) Application number: **98116477.5**

(22) Date of filing: **01.09.1998**

(54) Microcontroller, data processing system and task switching control method

Mikrokontroller, Datenverarbeitungssystem und Taskschaltungssteuerungsverfahren

Microcontrolleur, système de traitement de données et méthode de commande de changement de tâches

(84) Designated Contracting States:
DE FR GB NL

(30) Priority: **01.09.1997 JP 23562597**

(43) Date of publication of application:
31.03.1999 Bulletin 1999/13

(73) Proprietor: **Panasonic Corporation**
Kadoma-shi
Osaka 571-8501 (JP)

(72) Inventors:
- **Imanishi, Hiroshi**
  Kameoka-shi,
  Kyoto, 621-0867 (JP)
- **Araki, Toshiyuki**
  Ibaraki-shi,
  Osaka, 567-0825 (JP)

(74) Representative: **Grünecker, Kinkeldey,**
Stockmair & Schwanhäusser
Anwaltssozietät
Leopoldstrasse 4
80802 München (DE)

(56) References cited:
US-A- 4 914 570
US-A- 5 237 686
US-A- 5 655 146

- **MASAKI TOYOKURA ET AL: "A VIDEO DIGITAL SIGNAL PROCESSOR WITH A VECTOR-PIPELINE ARCHITECTURE" , IEEE INTERNATIONAL SOLID STATE CIRCUITS CONFERENCE, IEEE INC. NEW YORK, US, VOL. 35, PAGE(S) 72-73,248 XP000315769 ISSN: 0193-6530 * the whole document ***
- **TOYOKURA M. ET AL: 'A VIDEO DSP WITH A MACROBLOCK-LEVEL-PIPELINE AND A SIMD TYPE VECTOR-PIPELINE ARCHITECTURE FOR MPEG2 CODEC' IEEE JOURNAL OF SOLID-STATE CIRCUITS vol. 29, no. 12, 01 December 1994, pages 1474 - 1481, XP002917559**

EP 0 905 618 B1

Description

[0001] This invention relates to a multitasking microcontroller and to a data processing system employing such a multitasking microcontroller operable to control a plurality of hardware engines and it further relates to a method of controlling task switching.

[0002] A multitasking computer system consisting of plural processors (CPU) is described in US-B-4 914 570. A process is assigned to one CPU and an entry referring to that process is created in a run queue of said CPU. Each CPU selects a process from its run queue according to the priority of each process. Specific tasks, which can not be performed by the CPU a process is assigned to, are performed by a different CPU suitable for performing said tasks. For that purpose, the process is transferred to the run queue of that CPU. When the specific tasks are done, the process returns to the run queue of the original CPU.

[0003] US-A-5 655,146 describes a computer system comprising central processors and coexecutors for executing offloaded functions in accordance with requests from programs running on a central processor. The coexecutors execute the offloaded functions in parallel with the central processors executing non-offloaded functions. A host OS handles dynamic scheduling of the coexecutor's operation. Specifically, operation requests to the coexecutors are stored in queues and coexecutors are signaled to perform a requested operation. Completion signals are issued by the coexecutors in order to notify a program which issued the request and to process the next request in the queues.

[0004] Both prior art references use a conventional interrupt driven task management and software scheduling.

[0005] Multitasking microcontrollers have been known in the art. In a typical multitasking microcontroller, a built-in processor sequentially executes a plurality of tasks and a task timer therefore makes periodic issues of timer interruptions which request task switching to be made. Every time the processor accepts such a timer interruption, an interruption handling routine in the operating system (OS) is activated. The interruption handling routine performs the scheduling of tasks and the saving and restoring of resources.

[0006] Conventional multitasking microcontrollers have some drawbacks. For example, the scheduling of tasks is carried out using an interruption handling routine in a conventional multitasking microcontroller. This produces the problem that there is created much overhead at task switching time, therefore resulting in a drop in microcontroller performance. This is a serious problem for the applications which attach great importance to real-time processing such as image data encoding.

[0007] Such a technology is described in IEEE Journal of Solid-State Circuits vol. 29, No. 12, December 1994, p.1474 - 1481 for an MPEG2 CODEC.

[0008] It is an object of the present invention to provide an improved control of a pipelined data processing system.

[0009] This object is achieved by the features of claim 1.

[0010] Further embodiments are subject-matter of dependent claims.

[0011] In accordance with the present invention, a plurality of tasks are allocated to respective hardware engines. In such an environment, a task switching operation is controlled by the hardware scheduler on the basis of information representative of the allocation of the tasks to the hardware engines. Some of the hardware engines execute time critical processes and the other hardware engines do not. In accordance with the present invention, a relationship among the hardware engines is reflected in the execution priority of the tasks, which makes it possible to select a task to be run next in a short time without redetermining which of the hardware engines executes a time critical process at the time of task switching. In other words, there is created less overhead when task switching occurs. High-speed task switching is realized.

[0012] Undesirable dead time occurs in a time sharing method which carries out switching between tasks in response to an interruption periodically issued by a task timer, in view of which the present invention was made. Accordingly, the microcontroller of the present invention adopts an event-driven method capable of performing task switching in fast response to the occurrence of an event (i.e., an event of hardware engine execution termination). Each task can be in one of at least three states: a first state (the state of READY) representative of an execution wait status, a second state (the state of ACTIVE) representative of a running status, and a third state (the state of SLEEP) representative of an allocated hardware engine execution termination status. A task is in the state of ACTIVE when it uses the microcontroller and in the state of ACTIVE a hardware engine allocated to the task is controlled. A task is in the state of READY when it is not selected therefore waiting to be selected although it is ready to use the microcontroller. A task is in the state of SLEEP when it waits for a hardware engine allocated thereto to be execution-terminated (in other words, it is not ready to use the microcontroller). A task that has finished activating its allocated hardware engine makes a state transition from ACTIVE to SLEEP in response to a given instruction (the task_sleep instruction). When the execution of a certain hardware engine is terminated, a task allocated to that hardware engine makes a state transition from SLEEP to READY and a task under execution makes a state transition from ACTIVE to READY. Thereafter, a task having the highest execution priority in all tasks assuming the state of READY is selected as a task to be run next. The task thus selected makes a state transition from READY to ACTIVE.

[0013] If a plurality of register files are prepared in the microcontroller so that a plurality of hardware engines can use the register files as mutually independent working areas, there is created much less overhead at the

time of task switching because what is required to do at the task switching time is just saving processor resources such as program counter. A register file for storing a setting parameter common to a plurality of hardware engines can be prepared in the microcontroller.

BRIEF DESCRIPTION OF THE DRAWINGS

[0014]

FIGURE **1** is a block diagram showing the structure of an MPEG image encoder in accordance with the present invention.

FIGURE **2** is a block diagram showing details of the structure of a microcontroller of FIGURE **1**.

FIGURE **3** is a block diagram showing details of the structure of a task controller of FIGURE **2**.

FIGURE **4** is a conceptual diagram showing an association between cores and tasks in the MPEG image encoder of FIGURE **1**.

FIGURE **5** is a conceptual diagram showing the state transition of a task in the MPEG image encoder of FIGURE **1**.

FIGURE **6** is a timing diagram showing macroblock pipeline processing by the five cores of FIGURE **1**.

FIGURE **7** is a timing diagram showing the state transition of three tasks during a part period of FIGURE **6**.

DETAILED DESCRIPTION OF THE INVENTION

[0015] FIGURE **1** shows an MPEG (Moving Picture Experts Group) image encoder which is one of the data processing systems in accordance with the present invention. Referring to FIGURE **1**, the MPEG image encoder has a single microcontroller **101**, five different hardware engines (hereinafter called the cores) **111-115** together forming a macroblock pipeline, and three buffer memories **116-118**. The core **111** is a motion detector (MD). The core **112** is a motion compensator (MC). The core **113** is a discrete cosine transformer (DCT). The core **114** is a quantizer (Q). The core **115** is a variable length coder (VLC). All the cores **111-115** are controlled by the microcontroller **101**. **121** is image data to be encoded and **122** is encoded data representative of a result of the encoding operation. The microcontroller **101** sends an activation signal **123** to each core **111-115** and receives a termination signal **124** from each core **111-115**. The microcontroller **101** is allowed to individually communicate with each core **111-115** through signal lines **131-135**. Additionally, the microcontroller **101** provides a parameter common to the five cores **111-115** through a signal line **136**.

[0016] FIGURE **2** shows in detail the structure of the microcontroller **101**. The microcontroller **101** has a task controller **201** for the realization of multitasking, five core register files **211-215** for use by the five cores **111-115** as mutually independent working areas, a single common register file **216** for storing a setting parameter common to at least two of the five cores **111-115**, a single general-purpose register file **217** for use by the task controller **201** as a working area, a multiplier **221**, a shifter **222**, an arithmetic and logic unit (ALU) **223**, and a data memory **224**. **241** is an A bus. **242** is a B bus. **243** is a C bus. **231** is a signal line for connecting together the buses **241-243** and the task controller **201**. The task controller **201** provides the activation signal **123** and receives the termination signal **124**. Each of the register files **211-216** is connected between the C bus **243** and a corresponding one of the signal lines **131-136**. Additionally, each of the register files **211-216** has two outputs that are connected to the A bus **241** and to the B bus **242** respectively. The general-purpose register file **217** and the data memory **224** each have a single input that is connected to the C bus **243** and two outputs that are connected to the A bus **241** and to the B bus **242** respectively. The multiplier **221**, the shifter **222**, and the ALU **223** each have two inputs that are connected to the A bus **241** and to the B bus **242** respectively and a single output that is connected to the C bus **243**. A variation to the above can be made in which the placement of the five core register files **211-215** and the common register file **216** is omitted and the signal lines **131-136** extend directly form the C bus **243**.

[0017] In accordance with the MPEG image encoder of FIGURE **1**, image data processes proceed in units of macroblocks each containing 16×16 pixels. Firstly, the MD core **111** finds candidate motion vectors about the input image data **121**. In the MC core **112**, image differential data are found using the candidate motion vectors to select an optimal motion vector. Differential data with respect to the selected motion vector is discrete cosine transformed in the DCT core **113**, quantized in the Q core **114**, variable length coded in the VLC core **115** together with side information such as the motion vectors found, and finally provided as the encoded data **122**.

[0018] The above is discussed in detail with reference to FIGURE **2**. The task controller **201** first sets an operating parameter to the MD core register file **211** through the signal line **231**, the ALU **223**, and the C bus **243** and provides the activation signal **123** to make the MD core **111** active. The MD core **111** reads in the operating parameter from the MD core register file **211** through the signal line **131** and inputs the image data **121**. Upon termination of the execution of the MD core **111**, candidate motion vectors found in the MD core **111** are written into the MD core register file **211** through the signal line **131**, and the MD core **111** provides the termination signal **124**. In response to the termination signal **124**, the task controller **201** reads out the candidate motion vectors from the MD core register file **211**. Based on the candidate motion vectors, the task controller **201** computes an operating parameter for the MC core **112** by the use of the multiplier **221**, the shifter **222**, the ALU **223**, and the general-purpose register file **217**. The operating parameter is set to the MC core register file **212** and the MC core **112** is made active by the activation signal **123**. The MC

core **112** reads in the operating parameter from the MC core register file **212** through the signal line **132**. Thereafter, the MC core **112** finds image differential data. Upon termination of the execution of the MC core **112**, a sum of the image differential data is written into the MC core register file **212** by way of the signal line **132**, the image differential data are written into the buffer memory **116**, and the MC core **112** provides the termination signal **124**. In response to the termination signal **124**, the task controller **201** reads out the image differential data sum from the MC core register file **212**. Based on the image differential data sum, the task controller **201** selects an optimum motion vector from among the aforesaid candidate motion vectors through the use of the multiplier **221**, the shifter **222**, the ALU **223**, and the general-purpose register file **217**. An address indicative of the location of differential data corresponding to the optimum motion vector is set in the DCT core register file **213** and the DCT core **113** is made active by the activation signal **123**. Based on the address set in the DCT core register file **213**, the DCT core **113** reads out the differential data from the buffer memory **116** for DCT. Upon termination of the execution of the DCT core **113**, a result of the DCT operation is written into the buffer memory 117 and the DCT core **113** provides the termination signal **124**. The Q core **114** performs quantization and a result of the quantization operation is written into the buffer memory **118**. The VLC core **115** performs VLC and a result of the VLC operation is provided as the encoded data **122**. Some of the five cores **111-115** exchange the signals **123** and **124** with the microcontroller **101** a plurality of times per macroblock processing. The common register file **216** is used in cases such as when a common parameter for switching between MPEG1 and MPEG2 is pre-supplied to the five cores **111** and when a common parameter for designating a motion estimation mode is pre-supplied to the cores **111** and **112**.

[0019] Referring to FIGURE **3**, the structure of the task controller **201** is now described in detail. The task controller **201** has a processor **300**, a task management table **310**, and a scheduler **330**. The processor **300** is a RISC (reduced instruction set computer) processor capable of sequential execution of eight tasks at most. The processor **300** has a program counter (PC) **301** for generating instruction addresses, an instruction memory **302** for storing a program of a series of instructions, and an instruction decoder **303** for decoding instructions. The instruction decoder **303** sends the activation signal **123** to each core. The instruction decoder **303** is connected to resources for the execution of instructions, such as the multiplier **221**, the shifter **222** and the ALU **223**, through the signal line **231**. The task management table **310** is a circuit block for storing task management information. The task management table **310** has eight task management register files **320-327** that are associated with eight tasks from TASK0 to TASK7, respectively. The task management information includes state information (ST INFO) representative of the execution status of each task, priority information (PRI INFO) representative of the execution priority of each task, and core identification information (CID INFO) representative of the allocation of the tasks to the five cores **111-115**. Additionally, the task management table **310** has PC regions for the tasks for saving the processor's **300** resources (i.e., the contents of the PC **301**). Such a region is also used to save a flag concerning a result of the arithmetic operation of the ALU **223** (see FIGURE **2**). The scheduler **330** is a circuit block operable to allows the processor **300** to switch between tasks on the basis of the task management information stored in the task management table **310**. The scheduler **330** has a state controller **331**, a terminated core determination unit (TCDU) **332**, a priority encoder **333**, and a selector **334**. In response to the termination signal **124** sent from any one of the five cores **111-115** (i.e., a core the execution of which is terminated), the TCDU **322** identifies a task allocated to that execution-terminated core. Such identification is carried out with reference to the task management table **310** and a task number **362** representative of a result of the identification operation is communicated to the state controller **331**. The priority encoder **333** is a circuit block for selecting a task to be run next. Referring to the task management table **310**, the priority encoder **333** performs such a selection operation and a task number **361** representative of a result of the selection operation is communicated to the state controller **331** as well as to the selector **334**. The state controller **331** is a circuit block for updating the ST INFO stored in the task management table **310**. The selector **334** controls the restoration of resources to the processor **300**.

[0020] FIGURE **4** illustrates an association between the cores and the tasks in the MPEG image encoder of FIGURE **1**. Here, the microcontroller **101** executes six tasks **400-405**. The task **400** controls the five tasks **401-405** lower in hierarchy than the task **400** and is a main task (TASK0) for managing the entire encoding processing. The main task **400** is assigned no core. The task **401** is a motion detection task (TASK1) for controlling the allocated MD core **111**. The task **402** is a motion compensation task (TASK2) for controlling the allocated MC core **112**. The task **403** is a DCT task (TASK3) for controlling the allocated DCT core **113**. The task **404** is a Q task (TASK4) for controlling the allocated Q core **114**. Finally, the task **405** is a VLC task (TASK5) for controlling the allocated VLC core **115**.

[0021] Suppose here that the task management table **310** of FIGURE **3** stores task management information concerning at least six tasks (the six tasks **400-405**). Referring to FIGURE **3**, the PRI INFO is set according to a priority setting signal **342**. The CID INFO is set according to a core setting signal **343**. The priority setting signal **342** is sent to the task management table **310** from the instruction decoder **303** if the instruction decoder **303** decodes a priority setting instruction. On the other hand, the core setting signal **343** is sent to the task management table **310** from the instruction decoder **303** if the

instruction decoder **303** decodes a core setting instruction.

[0022] FIGURE **5** is a conceptual diagram showing the state transition of each task. Each task can be in one of four states, namely, the state of STOP representative of a suspended status, the state of READY representative of an execution wait status, the state of ACTIVE representative of a running status, and the state of SLEEP representative of an allocated hardware engine execution termination wait status. SLEEP cannot exist for TASK0. If the task controller **201** is reset, it will assume the state of STOP for all tasks. A task in the state of STOP is changed to READY by a task_ready instruction (in other words, a transition **501** is made). If a task in the state of READY is selected by the scheduler **330** when an event requesting for task switching to be made occurs, the task is changed to ACTIVE (in other words, a transition **511** is made), at which time a task which has been placed in the state of ACTIVE up to the moment is changed to READY by the scheduler **330** (in other words, a transition **522** is made). A task in the state of ACTIVE is executed by the processor **300**. A task in the state of ACTIVE can be changed to SLEEP by a task_sleep instruction (in other words, a transition **521** is made), alternatively it can be changed to STOP by a task_stop instruction (in other words, a transition **523** is made). A task in the state of SLEEP is changed to READY (in other words, a transition **531** is made) if the execution of a core allocated to that task is terminated.

[0023] Details of the operation of the task controller **201** of FIGURE **3** are described here. Task switching occurs if the instruction decoder **303** decodes the task_ready instruction, the task_sleep instruction, or the task_stop instruction. For example, when a certain task is run to finish setting an operating parameter for a core allocated to the task and activating the core, the state of the task is changed from ACTIVE to SLEEP by the task_sleep instruction. Additionally, task switching occurs upon termination of the execution of any one of the five cores **111-115**. The operating sequence of the task controller **201** at task switching time includes (1) activating the scheduler (SCHEDULER ACTIVATION), (2) saving the resources of a task under execution (RESOURCE SAVING), (3) selecting a task to be run next (TASK SELECTION), and (4) restoring the saved resources (RESOURCE RESTORATION).

[0024] Firstly, a task switching sequence on the basis of instructions is explained.

(A-1) SCHEDULER ACTIVATION

[0025] If the task_ready instruction, the task_sleep instruction, or the task_stop instruction is decoded, the instruction decoder **303** provides a state change signal **341**. The state change signal **341** is sent to the state controller **331**. As a result, the scheduler **330** is made active.

(A-2) RESOURCE SAVING

[0026] The state change signal **341** is also sent to the task management table **310** and the ST INFO is updated. At the same time, the value of the PC **301** of a task which has been executed up to the moment is saved in the task management table **310** through a signal line **344**.

(A-3) TASK SELECTION

[0027] The priority encoder **333** receives from the task management table **310** the ST INFO and the PRI INFO through a signal line **351** and through a signal line **352** respectively, to select a task having the highest priority of execution in all tasks that are in the state of READY as a task to be run next. The task number **361** indicative of a result of the task selection operation is communicated to the state controller **331** and to the selector **334**.

(A-4) RESOURCE RESTORATION

[0028] The state controller **331** sends to the task management table **310** a state change signal **364** according to the task number **361**. As a result, the ST INFO of the task selected by the priority encoder **333** is updated from READY to ACTIVE. The selector **334** reads out the PC of the task designated by the task number **361** from the task management table **310** through a signal line **353**, for forwarding onto a signal line **363**. As a result, the value of the PC of the task to be run next is set in the processor **300** and the execution of the task starts.

[0029] Next, a task switching sequence on the basis of core execution termination is now described.

(B-1) SCHEDULER ACTIVATION

[0030] If the execution of any one of the cores is terminated, the termination signal **124** is sent to the TCDU **332**. The TCDU **332** determines which of the cores is execution-terminated on the basis of the termination signal **124**. Further, the TCDU **332** reads out the CID INFO stored in the task management table **310** through a signal line **354** and determines which of the tasks is allocated to the execution-terminated core. The task number **362** indicative of a result of the determination operation by the TCDU **332** is communicated to the state controller **331** if the task in question is confirmed to be in the state of SLEEP from the ST INFO, in consequence of which the scheduler **330** is activated. The state controller **331** sends to the task management table **310** the state change signal **364** according to the task number **362**, as a result of which the ST INFO of the execution-terminated task is updated from SLEEP to READY. The scheduler **330** will not be activated if there exists no task allocated to the execution-terminated core.

(B-2) RESOURCE SAVING

[0031] The state controller **331** sends the state change signal **364** to the task management table **310** so that the ST INFO of a task that has been under execution up to the moment is updated from ACTIVE to READY. At the same time, the value of the PC **301** of the task is saved in the task management table **310**.

(B-3) TASK SELECTION

[0032] The priority encoder **333** receives the ST INFO and the PRI INFO from the task management table **310** thereby selecting a task having the highest priority of execution in all tasks in the state of READY as a task to be run next. The task number **361** indicative of a result of the task selection operation is communicated to the state controller **331** as well as to the selector **334**.

(B-4) RESOURCE RESTORATION

[0033] The state controller **331** sends to the task management table **310** the state change signal **364** according to the task number **361**. The ST INFO of the task selected by the priority encoder **333** is updated from READY to ACTIVE. The selector **334** reads out the PC of the task designated by the task number **361** from the task management table **310**, for forwarding to the processor **300**. As a result, the value of the PC of the task to be run next is set in the processor **300** and the execution of the task in question starts.

[0034] Referring now to FIGURE **6**, therein shown is macroblock pipeline processing by the five cores **111-115** of FIGURE **1**. The pipeline pitch is set at a maximum value of the time taken to process one macro block in each core, which means that there is a characteristic that in the individual pipeline pitches, there exists a core the execution of which is terminated earlier than the other cores. This therefore produces idle time and there is a characteristic that the length of such idle time varies depending on the image data. In the example of FIGURE **1**, an MPEG image encoder adaptive to the foregoing characteristics is implemented by means of task switching adopting an event driven method. The number of times a core is activated in an individual pipeline pitch period varies depending on the contents and data of processing carried out in the core. For instance, the DCT core **113** is activated once per pipeline pitch period. On the other hand, the MC core **112** is activated a plurality of times per pipeline pitch period depending on the data, since in the MC core **112** data of one macroblock is divided into luminance and chrominance components and processing is subdivided for the components and carried out.

[0035] FIGURE **7** shows the state transition of each of three tasks in a part period specified by broken line of FIGURE **6**. TASK0 is a main task for managing an entire encoding process, TASK1 is a task allocated to the MD core **111**, and TASK2 is a task allocated to the MC core **112** (see FIGURE **4**). In these three tasks, TASK1 has the highest execution priority. TASK2 has the second highest execution priority. TASK0 has the lowest execution priority. Suppose that at time t0, TASK1 is in the state of ACTIVE and TASK0 and TASK2 are in the state of READY.

[0036] FIGURE **7** shows that task switching occurs at each time t1-t7. In FIGURE **7**, At represents the overhead of one task switching operation. The description will be made in order. TASK1 makes the MD core **111** active prior to time t1. At time t1, the state of TASK1 is changed from ACTIVE to SLEEP by the task_sleep instruction. At this point in time, although TASK0 and TASK2 are in the state of READY, it is TASK2 that is allowed to make a transition from READY to ACTIVE, since TASK2 has priority of execution over that of TASK0. TASK2 makes the MC core **112** active. Then, at time t2, the state of TASK2 is changed from ACTIVE to SLEEP by the task_sleep instruction. At this point in time, only TASK0 assumes the state of READY. TASK0 therefore makes a state transition from READY to ACTIVE. At time t3, the state of TASK2 is changed from SLEEP to READY by the execution termination of the MC core **112** and the state of TASK0 (which has been in the state of ACTIVE up to the moment) moves to READY. At this point in time, although TASK0 and TASK2 are in the state of READY, it is TASK2 that is allowed to make a state transition from READY to ACTIVE, since TASK2 has priority of execution over that of TASK0. TASK2 again activates the MC core **112**. At time t4, the state of TASK2 is changed from ACTIVE to SLEEP by the task_sleep instruction, at which point in time only TASK0 is in the state of READY. TASK0 therefore makes a state transition from READY to ACTIVE. At time t5, the state of TASK1 is changed from SLEEP to READY by the execution termination of the MD core **111** and the state of TASK0 (which has been in the state of ACTIVE up to the moment) moves to READY. At this point in time, although TASK0 and TASK1 are in the state of READY, it is TASK1 that is allowed to make a state transition from READY to ACTIVE, since TASK1 has priority of execution over that of TASK0. At time t6, the state of TASK2 is changed from SLEEP to READY by the execution termination of the MC core **112** and the state of TASK1 (which has been in the state of ACTIVE up to the moment) moves to READY. At this point in time, although TASK0, TASK1, and TASK2 are all in the state of READY, it is TASK1 that is allowed to return to ACTIVE from READY, since TASK1 has the highest execution priority in all the tasks. TASK1 again makes the MD core **111** active. At time t7, the state of TASK1 is changed from ACTIVE to SLEEP by the task_sleep instruction. At this point in time, although TASK0 and TASK2 are in the state of READY, it is TASK2 that is allowed to make a state transition from READY to ACTIVE, since TASK2 has priority of execution over that of TASK0.

[0037] In cases where task switching, based on a time sharing method utilizing a conventional interruption han-

dling routine, is adopted, the overhead of one task switching operation amounts to ten-odd machine cycles. On the other hand, for the case of task switching based on an event driven method according to the present invention, the overhead, At, of FIGURE **7** is only several machine cycles. Taking into account the fact that task switching occurs twenty-odd times at most in an individual macropipeline pitch period, the difference in overhead between the foregoing methods further increases. Reduction in the overhead achieved by the present invention makes it possible to achieve also a reduction in the pipeline pitch. In other words, it is possible to achieve high-speed encoding of image data.

[0038] As described above, high-speed task switching can be realized in the MPEG image encoder of FIGURE **1**. If a task that executes a time critical process is assigned higher priority of execution, this guarantees normal image encoding processing. When the execution of a core is terminated, both the state of a task allocated to the execution-terminated core and the state of a task which has been under execution up to the moment are first changed to READY, and a task having the highest priority of execution in all tasks in the state of READY is selected as one to be run next. This can provide a simplified internal structure to the priority encoder **333**. Additionally, it is possible to independently describe programs for tasks. This not only improves efficiency in programming but also provides advantages over debugging.

[0039] The present invention is applicable to other data processing systems including an image decoder. In the foregoing description, each hardware engine (core) is assigned one task. There may be a core that is assigned no task. Additionally, there may be a core that is assigned a plurality of tasks. The same task is never simultaneously allocated to a plurality of cores.

Claims

1. A data processing system comprising:

a plurality of hardware engines (111-115) forming a macro block pipeline, each of the hardware engines being assigned one task in a plurality of repetitive operations of encoding or decoding a data stream consisting of consecutive blocks of data, and
a microcontroller (101) for controlling said plurality of hardware engines, said microcontroller adopting an event-driven method capable of performing task switching in fast response to the occurrence of an event, said microcontroller including:

a processor (300) having a program counter (301) for sequentially executing said plurality of tasks;
a task management table (310) for storing task management information, including:

(i) state information representative of the execution status of each task,
(ii) priority information representative of the execution priority of each task,
(iii) allocation information representative of the allocation of said plurality of tasks to said hardware engines, and
(iv) program counter values for each task;

a plurality of register files (211-216) which can be used by said hardware engines as mutually independent working areas, and
a hardware scheduler (330) for allowing said processor to switch between said tasks without an interrupt handling routine, on the basis of said task management information,
said hardware scheduler (330) including:

a) a determination unit (332) for identifying a task allocated to an execution-terminated hardware engine (111-115) on the basis of said task management information, when the execution of any one of said plurality of said hardware engines (111-115) is terminated;
b) a state controller (331) which performs a function of updating of state information, upon being activated by said determination unit (332), and
c) a selector (334) that reads out a program counter value for the task to be run next in the processor (300) said task being determined by said state information and priority information and said value being set in the processor to start the execution of that task.

2. The data processing system according to claim 1, wherein each of said tasks is in one of the following states:

a first state (READY-state) representative of an execution wait status,
a second state (ACTIVE-state) representative of a running status, and
a third state (SLEEP-state) representative of a wait status awaiting the termination of execution of a hardware engine (111 - 115) allocated thereto.

3. The data processing system according to claim 2, wherein when during execution of a task of said plurality of tasks said microprocessor (101) activates a hardware engine (111 - 115) allocated to said task under execution before decoding a given instruction, said microprocessor (101) performs a function of up-

dating said state information so that said task makes a state transition from said second state (ACTIVE-state) to said third state (SLEEP-state).

4. The data processing system according to claim 2, wherein:

the state controller (331) performs the function of updating said state information when a task makes a state transition from said third state (SLEEP-state) to said first state (READY-state).

5. The data processing system according to claim 4, wherein said state controller (331) performs, upon being activated by said determination unit (332), a function of updating said state information so that a task under execution makes a state transition from said second state (ACTIVE-state) to said first state (READY-state).

6. The data processing system according to claim 2, wherein said scheduler (330) further includes a priority encoder (333) for selecting, on the basis of said task management information, a task having the highest execution priority in all tasks that are in said first state (READY-state) as a task to be run next.

7. The data processing system according to claim 6, wherein said state controller (331) further performs a function of updating said state information so that said task selected by said priority encoder (333) makes a state transition from said first state (READY-state) to said second state (ACTIVE-state).

8. The data processing system according to any of claims 1 - 7, wherein said task management table (310) has a region in which to save resources of said processor (300) concerning a task that was run prior to the occurrence of a task switching.

9. The data processing system according to any of claims 1 - 8, further comprising a register file (216) used to store a setting parameter common to at least two of said plurality of hardware engines.

10. The data processing system according to any of claims 1 - 9, wherein each of said plurality of hardware engines (111 - 115) is a subprocessing core for MPEG image data encoding.

Patentansprüche

1. Datenverarbeitungssystem, das umfasst:

eine Vielzahl von Hardware-Engines (111-115), die eine Makroblock-Pipeline bilden, wobei jeder der Hardware-Engines eine Aufgabe einer Vielzahl sich wiederholender Operationen des Codierens oder Decodierens eines Datenstroms zugewiesen ist, der aus aufeinanderfolgenden Blöcken von Daten besteht, und
einen Mikrocontroller (101) zum Steuern der Vielzahl von Hardware-Engines, wobei der Mikrocontroller ein ereignisgesteuertes Verfahren anwendet, mit dem Aufgabenwechsel in schneller Reaktion auf das Auftreten eines Ereignisses möglich ist, und der Mikrocontroller enthält:

einen Prozessor (300) mit einem Programmzähler (301) zum sequenziellen Ausführen der Vielzahl von Aufgaben;
eine Aufgabenverwaltungstabelle (310) zum Speichern von Aufgabenverwaltungsinformationen, die einschließen:

I) Zustandsinformationen, die repräsentativ für den Ausführungszustand jeder Aufgabe sind,
II) Prioritätsinformationen, die repräsentativ für die Ausführungspriorität jeder Aufgabe sind,
III) Zuordnungsinformationen, die repräsentativ für die Zuordnung der Vielzahl von Aufgaben zu den Hardware-Engines sind, und
IV) Programmzählerwerte für jede Aufgabe;

eine Vielzahl von Registerdateien (211-216), die von den Hardware-Engines als voneinander unabhängige Arbeitsbereiche verwendet werden können, und
einen Hardware-Scheduler (330), der es dem Prozessor ermöglicht, auf Basis der Aufgabenverwaltungsinformationen ohne eine Interrupt-Abwicklungsroutine zwischen den Aufgaben zu wechseln,

wobei der Hardware-Scheduler (330) enthält:

a) eine Feststelleinheit (332) zum Identifizieren einer Aufgabe, die einer Hardware-Engine (111-115), deren Ausführung beendet ist, auf Basis der Aufgabenverwaltungsinformationen zugewiesen wird, wenn die Ausführung einer der Vielzahl der Hardware-Engines (111-115) beendet ist;
b) eine Zustands-Steuereinheit (331), die eine Funktion des Aktualisierens von Zustandsinformationen erfüllt, wenn sie durch die Feststelleinheit (332) aktiviert wird, und
c) eine Auswähleinrichtung (334), die einen Programmzählerwert für die als nächstes in dem Prozessor (300) auszuführende Aufgabe ausliest, wobei die Aufgabe durch die Zustandsin-

formationen und Prioritätsinformationen bestimmt wird und der Wert in dem Prozessor eingestellt wird, um mit der Ausführung dieser Aufgabe zu beginnen.

2. Datenverarbeitungssystem nach Anspruch 1, wobei sich jede der Aufgaben in einem der folgenden Zustände befindet:

einem ersten Zustand (READY-Zustand), der repräsentativ für einen Ausführungs-Wartezustand ist,
einem zweiten Zustand (ACTIVE-Zustand), der repräsentativ für einen Ausführungszustand ist, und
einem dritten Zustand (SLEEP-Zustand), der repräsentativ für einen Wartezustand ist, in dem auf die Beendigung von Ausführung einer ihr zugeordneten Hardware-Engine (111-115) gewartet wird.

3. Datenverarbeitungssystem nach Anspruch 2, wobei, wenn während der Ausführung einer Aufgabe der Vielzahl von Aufgaben der Mikroprozessor (101) eine Hardware-Engine (111-115) aktiviert, die der Aufgabe zugeordnet ist, die ausgeführt wird, bevor ein gegebener Befehl decodiert wird, der Mikroprozessor (101) eine Funktion des Aktualisieren der Zustandsinformationen erfüllt, so dass die Aufgabe einen Zustandsübergang von dem zweiten Zustand (ACTIVE-Zustand) zu dem dritten Zustand (SLEEP-Zustand) vollzieht.

4. Datenverarbeitungssystem nach Anspruch 2, wobei:

die Zustands-Steuereinheit (331) die Funktion des Aktualisierens der Zustandsinformationen erfüllt, wenn eine Aufgabe einen Zustandsübergang von dem dritten Zustand (SLEEP-Zustand) zu dem ersten Zustand (READY-Zustand) vollzieht.

5. Datenverarbeitungssystem nach Anspruch 4, wobei die Zustands-Steuereinheit (331), wenn sie durch die Feststelleinheit (332) aktiviert wird, eine Funktion des Aktualisierens der Zustandsinformation so erfüllt, dass eine Aufgabe, die ausgeführt wird, einen Zustandsübergang von dem zweiten Zustand (ACTIVE-Zustand) zu dem ersten Zustand (READY-Zustand) vollzieht.

6. Datenverarbeitungssystem nach Anspruch 2, wobei der Scheduler (330) des Weiteren eine Prioritäts-Codiereinrichtung (333) enthält, mit der auf Basis der Aufgabenverwaltungsinformationen eine Aufgabe mit der höchsten Ausführungspriorität von allen Aufgaben, die sich in dem ersten Zustand (READY-Zustand) befinden, als eine als nächste auszuführende Aufgabe ausgewählt wird.

7. Datenverarbeitungssystem nach Anspruch 6, wobei die Zustands-Steuereinheit (331) des Weiteren eine Funktion des Aktualisierens der Zustandsinformationen so erfüllt, dass die durch die Prioritäts-Codiereinrichtung (333) ausgewählte Aufgabe einen Zustandsübergang von dem ersten Zustand (READY-Zustand) zu dem zweiten Zustand (ACTIVE-Zustand) vollzieht.

8. Datenverarbeitungssystem nach einem der Ansprüche 1 - 7, wobei die Aufgabenverwaltungstabelle (310) einen Bereich hat, in dem Ressourcen des Prozessors (300) gespeichert werden, die eine Aufgabe betreffen, die vor dem Auftreten eines Aufgabenwechsels ausgeführt wurde.

9. Datenverarbeitungssystem nach einem der Ansprüche 1 - 8, das des Weiteren eine Registerdatei (216) umfasst, die verwendet wird, um einen Einstellungsparameter zu speichern, der wenigstens zwei der Vielzahl von Hardware-Engines gemeinsam ist.

10. Datenverarbeitungssystem nach einem der Ansprüche 1 - 9, wobei jede der Vielzahl von Hardware-Engines (111-115) ein Teilverarbeitungs-Kern für MPEG-Bilddaten-Codierung ist.

Revendications

1. Système de traitement de données comprenant:

une pluralité de moteurs matériels (111-115) formant une architecture pipeline au niveau des macroblocs, en attribuant une tâche à chacun des moteurs matériels dans une pluralité d'opérations répétitives consistant à coder ou décoder un flux de données de blocs consécutifs de données, et
un microcontrôleur (101) destiné à commander ladite pluralité de moteurs matériels, ledit microcontrôleur adoptant un procédé événementiel capable d'exécuter une commutation de tâche en réponse rapide à l'occurrence d'un événement, ledit microcontrôleur incluant:

un processeur (300) ayant un compteur de programmes (301) pour exécuter séquentiellement ladite pluralité de tâches;
un tableau de gestion de tâches (310) destiné à stocker des informations concernant la gestion de tâches, incluant:

(i) des informations d'état qui représentent le statut d'exécution de chaque tâ-

che,
(ii) des informations de priorité qui représentent la priorité d'exécution de chaque tâche,
(iii) des informations d'attribution qui représentent l'attribution de ladite pluralité de tâches auxdits moteurs matériels, et
(iv) des valeurs du compteur de programmes pour chaque tâche;

une pluralité de fichiers de registre (211-216) qui peuvent être utilisés par lesdits moteurs matériels en tant que zones de travail mutuellement indépendantes, et
un ordonnanceur matériel (330) destiné à permettre audit processeur de basculer entre lesdites tâches sans un sous-programme de gestion d'interruption, sur la base desdites informations concernant la gestion de tâches,
ledit ordonnanceur matériel (330) incluant:

(a) une unité de détermination (332) pour identifier une tâche attribuée à un moteur matériel (111-115) à exécution achevée sur la base desdites informations concernant la gestion de tâches, lorsque l'exécution de l'une quelconque de ladite pluralité desdits moteurs matériels (111-115) est achevée.
(b) une unité de commande d'état (331) qui effectue une fonction de mise à jour des informations d'état, à son activation par ladite unité de détermination (332), et
c) un sélecteur (334) qui lit une valeur du compteur de programmes pour la tâche à exécuter par la suite dans le processeur (300), ladite tâche étant déterminée par lesdites informations d'état et lesdites informations de priorité et ladite valeur étant établie dans le processeur pour entamer l'exécution de cette tâche.

2. Système de traitement de données selon la revendication 1, dans lequel chacune desdites tâches est dans l'un des états suivants:

un premier état (état-PRÊT) qui représente un statut d'attente de l'exécution,
un deuxième état (état-ACTIF) qui représente un état d'exécution, et
un troisième état (état-DESACTIVE) qui représente un statut d'attente qui attend l'achèvement de l'exécution d'un moteur matériel (111-115) qui lui est attribué.

3. Système de traitement de données selon la revendication 2, dans lequel lors de l'exécution d'une tâche de ladite pluralité de tâches ledit microprocesseur (101) active un moteur matériel (111-115) attribué à ladite tâche en cours d'exécution avant de décoder une instruction donnée, ledit microprocesseur (101) effectue une fonction de mise à jour desdites informations d'état de sorte que ladite tâche réalise une transition d'états dudit deuxième état (état-ACTIF) audit troisième état (état-DESACTIVE).

4. Système de traitement de données selon la revendication 2, dans lequel:

l'unité de commande d'état (331) effectue la fonction de mise à jour desdites informations d'état lorsqu'une tâche réalise une transition d'états dudit troisième état (état-DESACTIVE) audit premier état (état-PRÊT).

5. Système de traitement de données selon la revendication 4, dans lequel ladite unité de commande d'état (331) effectue, à son-activation par ladite unité de détermination (332), une fonction de mise à jour desdites informations d'état de sorte qu'une tâche en cours d'exécution réalise une transition d'états dudit deuxième état (état-ACTIF) audit premier état (état-PRÊT).

6. Système de traitement de données selon la revendication 2, dans lequel ledit ordonnanceur (330) inclut en plus un codeur de priorité (333) pour sélectionner, sur la base desdites informations concernant la gestion de tâches, une tâche ayant la priorité d'exécution la plus élevée parmi toutes les tâches qui se trouvent dans ledit premier état (état-PRÊT) en tant que tâche à exécuter par la suite.

7. Système de traitement de données selon la revendication 6, dans lequel ladite unité de commande d'état (331) effectue en plus une fonction de mise à jour desdites informations d'état de sorte que ladite tâche sélectionnée par ledit codeur de priorité (333) réalise une transition d'états dudit premier état (état-PRÊT) audit deuxième état (état-ACTIF).

8. Système de traitement de données selon l'une des revendications 1-7, dans lequel ledit tableau de gestion de tâches (310) a une région dans laquelle l'on sauvegarde les ressources dudit processeur (300) concernant une tâche qui a été exécutée avant l'occurrence d'une commutation de tâches.

9. Système de traitement de données selon l'une des revendications 1-8, comprenant en plus un fichier de registre (216) utilisé pour stocker un paramètre de réglage commun à au moins deux moteurs parmi ladite pluralité de moteurs matériels.

10. Système de traitement de données selon l'une des

revendications 1-9, dans lequel chacun de ladite pluralité de moteurs matériels (111-115) est un noyau de sous-traitement pour un codage de données d'images MPEG.

Fig. 1

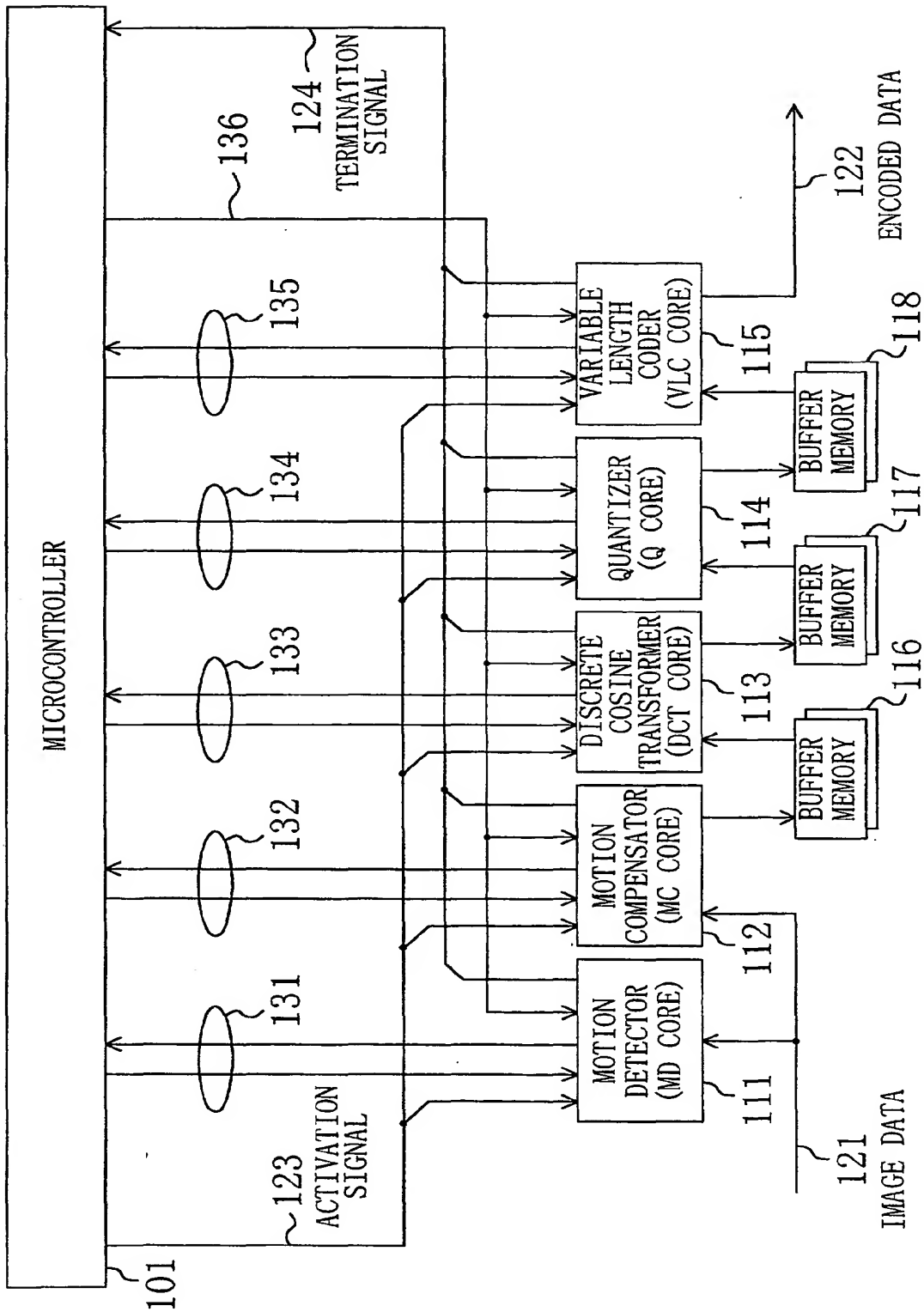

MICROCONTROLLER
101
123
ACTIVATION SIGNAL
131
132
133
134
135
136
124
TERMINATION SIGNAL
MOTION DETECTOR (MD CORE)
MOTION COMPENSATOR (MC CORE)
DISCRETE COSINE TRANSFORMER (DCT CORE)
QUANTIZER (Q CORE)
VARIABLE LENGTH CODER (VLC CORE)
111
112
113
114
115
BUFFER MEMORY
BUFFER MEMORY
BUFFER MEMORY
116
117
118
121
IMAGE DATA
122
ENCODED DATA

Fig. 2

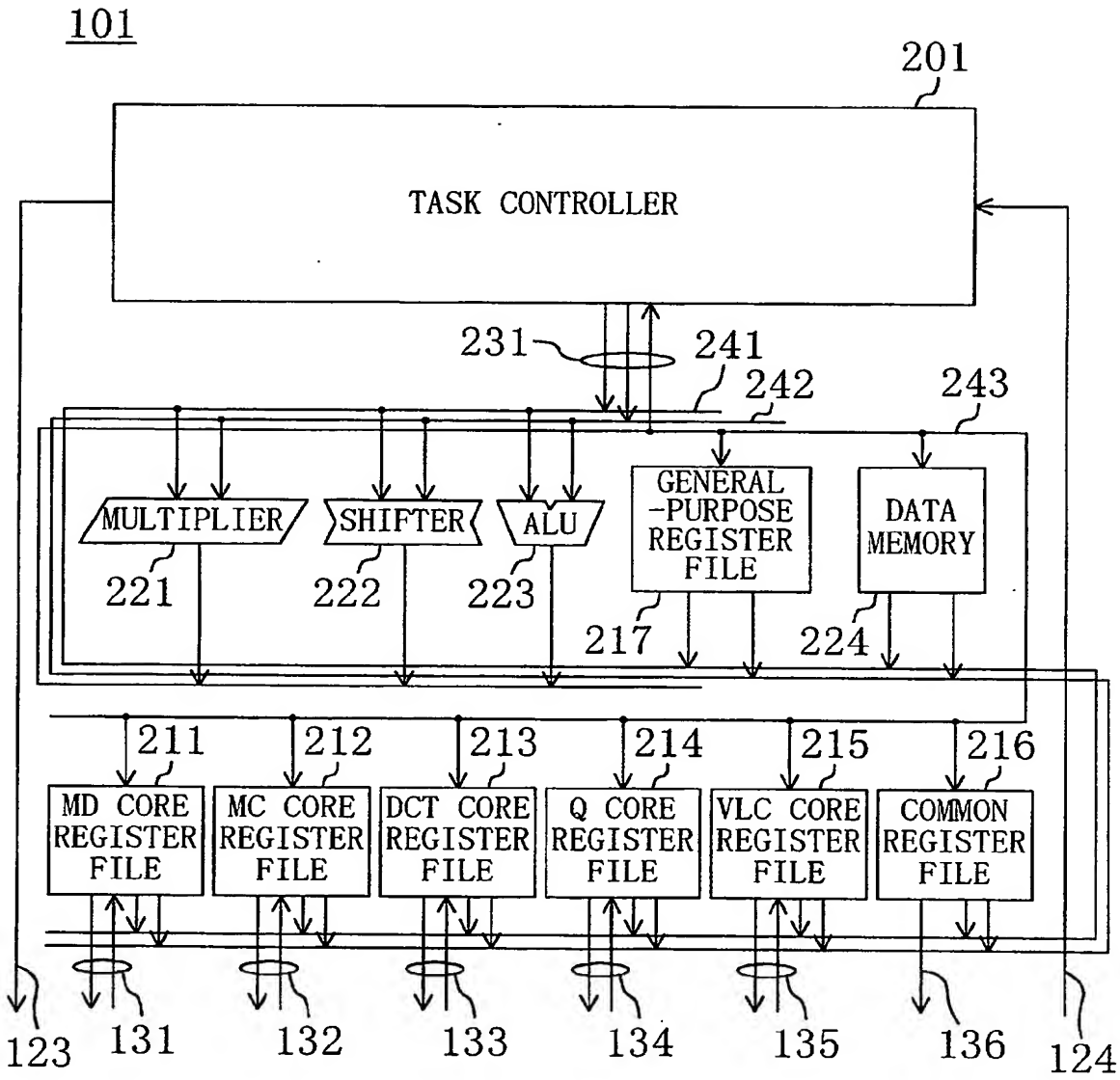

101
201
TASK CONTROLLER
231
241
242
243
MULTIPLIER
SHIFTER
ALU
GENERAL -PURPOSE REGISTER FILE
DATA MEMORY
221
222
223
217
224
211
212
213
214
215
216
MD CORE REGISTER FILE
MC CORE REGISTER FILE
DCT CORE REGISTER FILE
Q CORE REGISTER FILE
VLC CORE REGISTER FILE
COMMON REGISTER FILE
123
131
132
133
134
135
136
124

Fig. 3

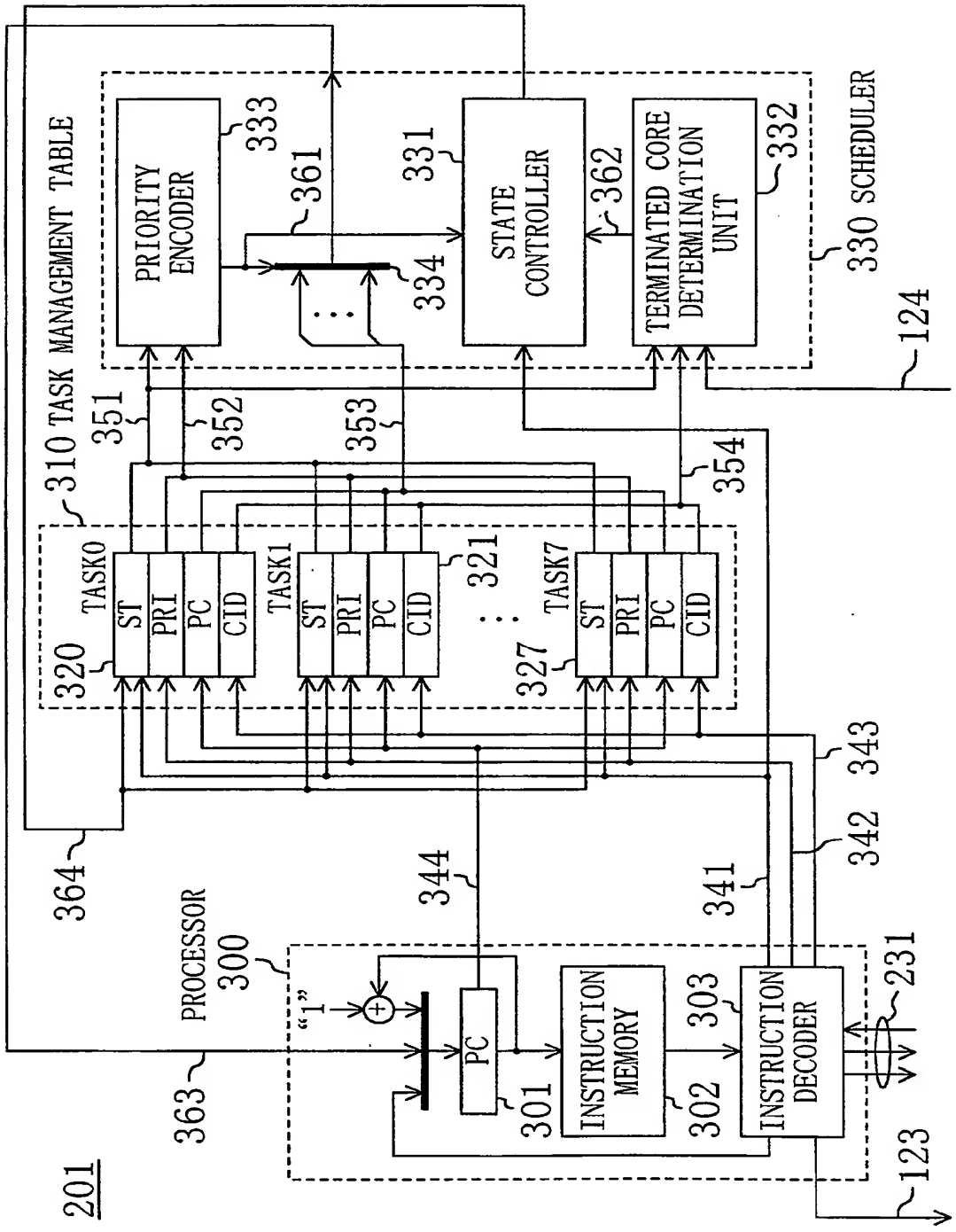

201
PROCESSOR
300
310 TASK MANAGEMENT TABLE
320
TASK0
TASK1
TASK7
ST
PRI
PC
CID
321
327
PRIORITY ENCODER
333
361
331
STATE CONTROLLER
362
TERMINATED CORE DETERMINATION UNIT
332
330 SCHEDULER
334
124
351
352
353
354
364
344
341
342
343
"1"
PC
301
INSTRUCTION MEMORY
302
303
INSTRUCTION DECODER
231
363
123

Fig. 4

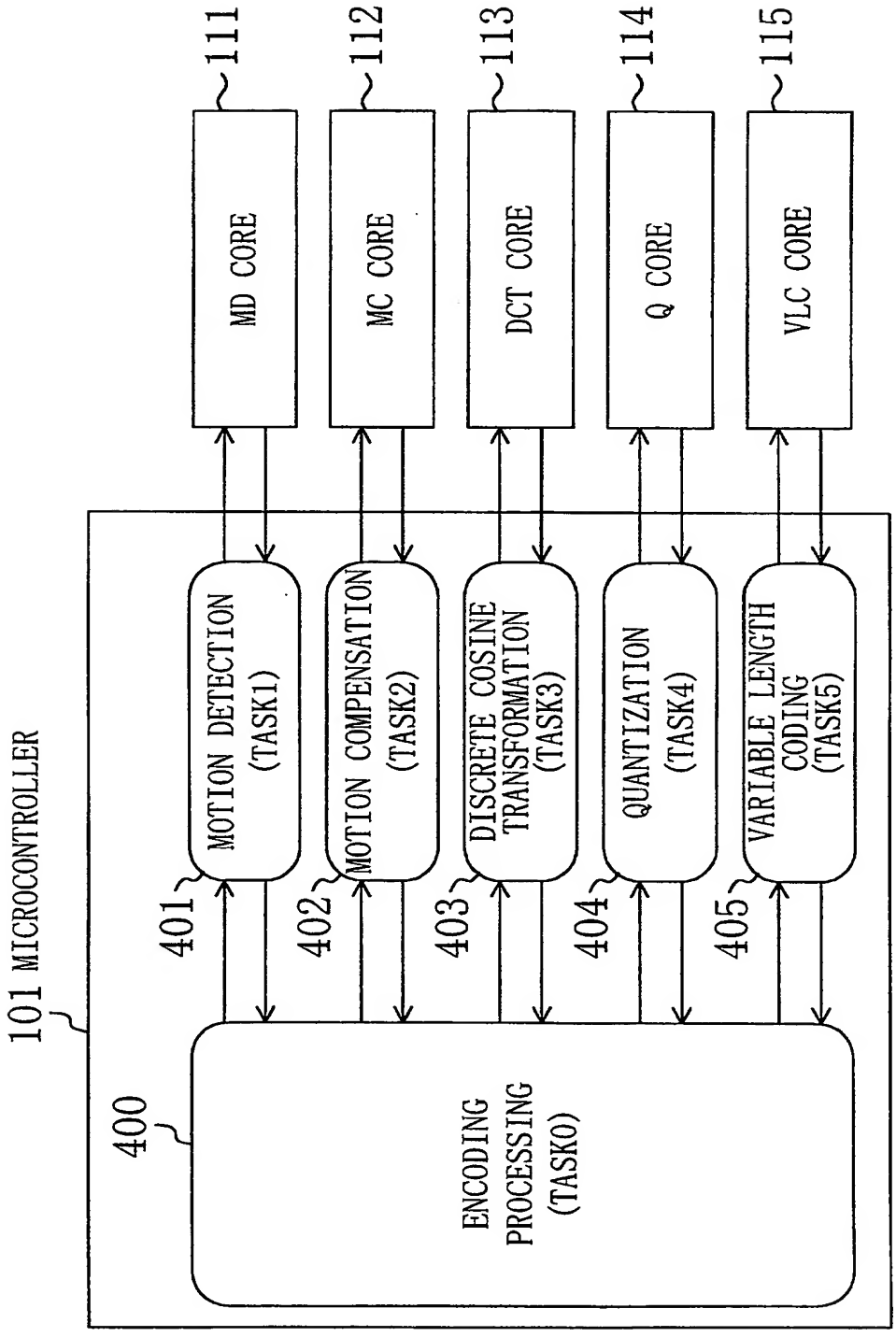

101 MICROCONTROLLER
400
ENCODING PROCESSING (TASK0)
401
MOTION DETECTION (TASK1)
402
MOTION COMPENSATION (TASK2)
403
DISCRETE COSINE TRANSFORMATION (TASK3)
404
QUANTIZATION (TASK4)
405
VARIABLE LENGTH CODING (TASK5)
111
MD CORE
112
MC CORE
113
DCT CORE
114
Q CORE
115
VLC CORE

Fig.5

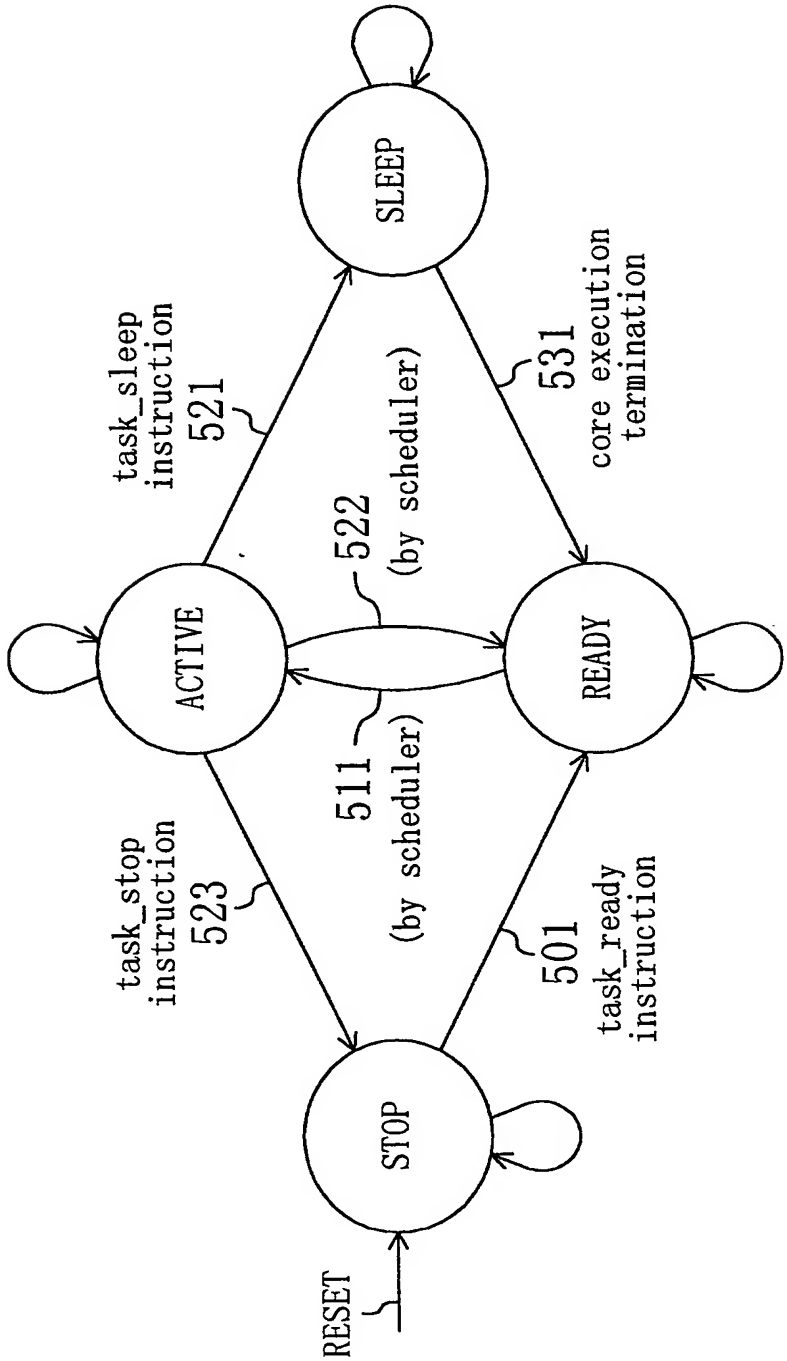

SLEEP
ACTIVE
READY
STOP
RESET
task_sleep instruction
521
522
(by scheduler)
531
core execution termination
511
(by scheduler)
task_stop instruction
523
501
task_ready instruction

Fig. 6

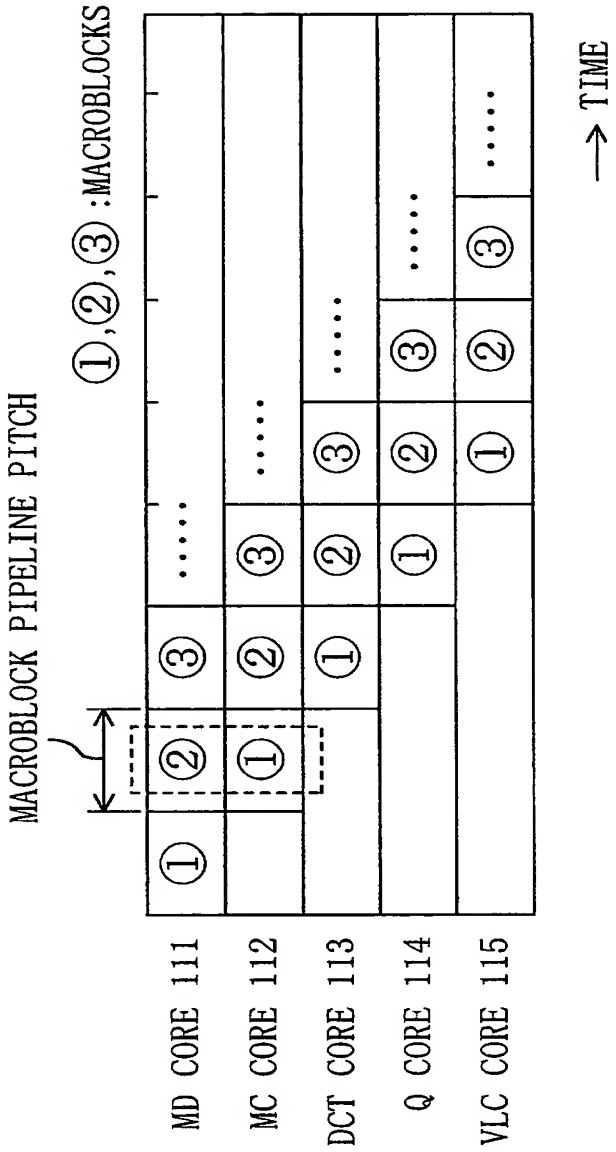
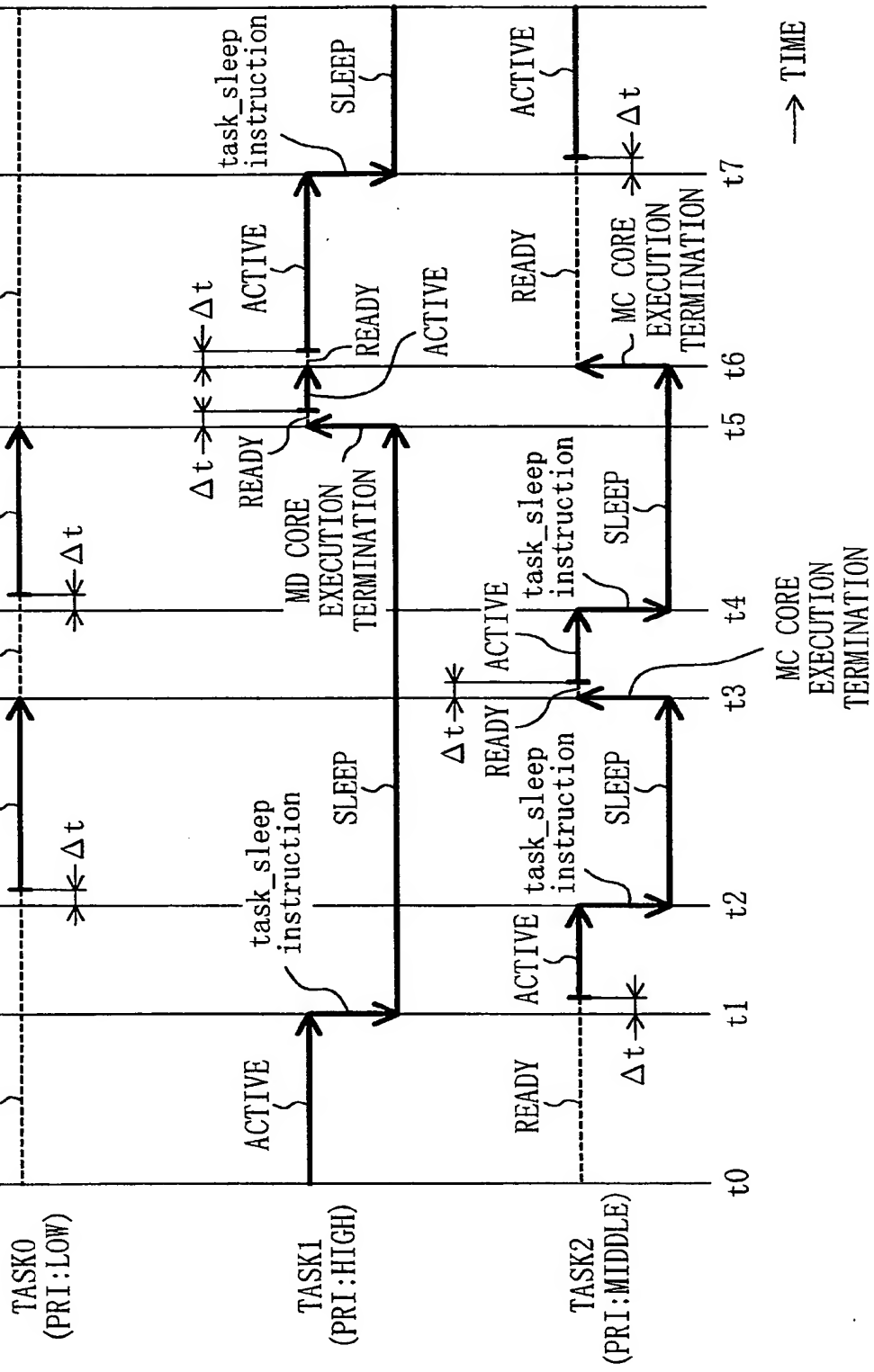

①,②,③ :MACROBLOCKS
MACROBLOCK PIPELINE PITCH
MD CORE 111
MC CORE 112
DCT CORE 113
Q CORE 114
VLC CORE 115
TIME

Fig. 7


Δt:OVERHEAD
TASK0 (PRI:LOW)
TASK1 (PRI:HIGH)
TASK2 (PRI:MIDDLE)
READY
ACTIVE
SLEEP
task_sleep instruction
MD CORE EXECUTION TERMINATION
MC CORE EXECUTION TERMINATION
Δt
t0
t1
t2
t3
t4
t5
t6
t7
→ TIME

REFERENCES CITED IN THE DESCRIPTION

This list of references cited by the applicant is for the reader's convenience only. It does not form part of the European patent document. Even though great care has been taken in compiling the references, errors or omissions cannot be excluded and the EPO disclaims all liability in this regard.

Patent documents cited in the description

- US 4914570 B **[0002]**
- US 5655146 A **[0003]**

Non-patent literature cited in the description

- *IEEE Journal of Solid-State Circuits,* December 1994, vol. 29 (12), 1474-1481 **[0007]**